TWOTOWERS: A TOOL INTEGRATING FUNCTIONAL AND PERFORMANCE ANALYSIS OF CONCURRENT SYSTEMS

M. Bernardo[1], W.R. Cleaveland[2], S.T. Sims[3], W.J. Stewart[2]

[1]Università di Bologna, Dipartimento di Scienze dell'Informazione
Mura A. Zamboni 7, 40127 Bologna, Italy
bernardo@cs.unibo.it

[2]North Carolina State University, Department of Computer Science
Raleigh, NC 27695-7534, USA
rance@eos.ncsu.edu, billy@eos.ncsu.edu

[3]Naval Research Lab, Center for High Assurance Computer Systems
Washington, DC 20375, USA
sims@itd.nrl.navy.mil

Abstract: We present TwoTowers, a tool for analyzing functional and performance properties of concurrent systems expressed as terms in the stochastically timed reward process algebra EMPA$_\mathrm{r}$. TwoTowers builds on two existing tools, CWB-NC and MarCA, that have been retargeted to carry out functional and performance analysis (respectively) of EMPA$_\mathrm{r}$ system specifications. As an example, we describe the application of TwoTowers to the Lehmann-Rabin randomized distributed algorithm for the dining philosopher problem.

INTRODUCTION

The desirability of taking account of the performance aspects of a system in the early stages of its design has been widely recognized [23, 10, 13, 7]. Nevertheless, it often happens that a concurrent system is tested for efficiency only after it has been fully designed and tested for functionality. This results in two problems. On the one hand, the detection of poor performance causes the system to be designed again, so the cost of the project increases. On the other hand, since the performance-related analysis is done on a model of a system extracted from the implementation while the functionality-related analysis is usually performed on a model derived from a system design, great care must be taken to ensure that these models are consistent with one another, i.e. do reflect (different aspects of) the same system.

In order to solve the two problems above, in [3, 4] a two-phase approach for modeling and analyzing both functional and performance characteristics of concurrent systems has been proposed, which is based on stochastically timed process algebras and stochastically timed Petri nets in order to exploit their complementary advantages:

- The first phase requires the designer to specify the concurrent system as a term in the stochastically timed process algebra. Because of compositionality, the designer is allowed to develop the algebraic representation of the system in a modular way: every subsystem can be modeled separately, then these models can be put together through the operators of the algebra. The analysis of the algebraic term is then carried out on its underlying integrated interleaving semantic model, which is a transition system labeled with both action types and durations. The integrated interleaving semantic model can be analyzed as a whole by a notion of integrated equivalence or projected on a functional semantic model and a performance semantic model that can be analyzed by means of existing tools.

- The second phase consists of deriving from the algebraic representation of the system an equivalent representation in the form of a stochastically timed Petri net. The net representation turns out to be useful whenever a less abstract representation is required highlighting dependencies, conflicts, and synchronizations among system activities, and helpful in detecting some properties that can be easily checked only in a distributed setting. Additionally, the net representation is usually more compact than the integrated interleaving semantic model resulting from the algebraic representation, since concurrency is kept explicit instead of being simulated by alternative computations obtained by interleaving actions of concurrent components. The functional and performance analysis of the net representation can then be undertaken using existing tools.

Since the two phases above are complementary, the choice between them is made according to the adequacy of the related representation with respect to the analysis of the system under consideration and the availability of the corresponding tools. In any case, the designer is suggested to start with an algebraic representation of the system in order to take advantage of compositionality of algebras and avoid graphical complexity of nets.

This paper describes our efforts to provide tool support for the first phase of this approach through the implementation of a tool called TwoTowers. To use TwoTowers, the designer first specifies the concurrent system as an algebraic term. The algebra adopted in the tool is $\mathrm{EMPA}_{\mathrm{r}}$ [1], an extension of the stochastically timed process algebra EMPA [5] allowing for the specification of performance measures based on rewards, which augments the expressive power of a classical process algebra with additional features for expressing probabilities, priorities, and timing characteristics of systems. Because $\mathrm{EMPA}_{\mathrm{r}}$ supports compositional modeling, the designer may develop the algebraic representation of the system in a modular way; every subsystem can be modeled separately, then combined using the appropriate operators. Another useful feature of $\mathrm{EMPA_r}$ is that many functional properties of a system may be examined independently of performance properties and vice versa. Our implementation of TwoTowers exploits this fact by using two existing tools (that we have retargeted to $\mathrm{EMPA_r}$) for the analysis of these types of properties. CWB-NC [9] provides a variety of different types of analysis (e.g. model checking, equivalence checking, preorder checking, reachability analysis) of the functional behavior of systems, while MarCA [21] conducts stationary and transient performance analysis, as depicted in Fig. 1.1.

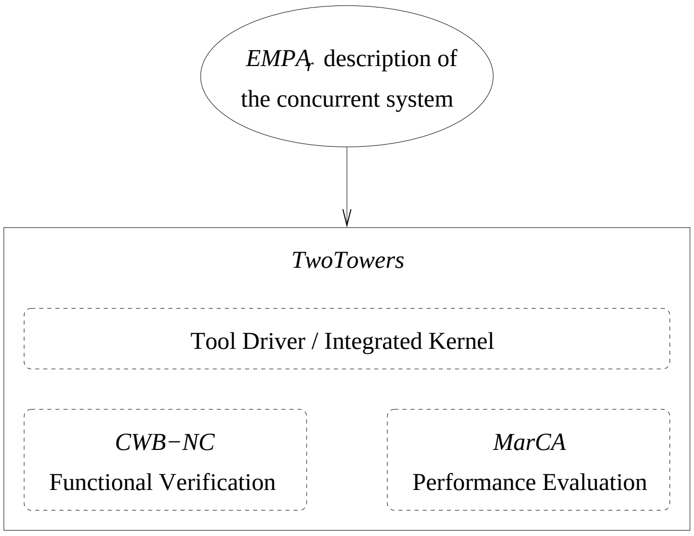
EMPA_r description of the concurrent system
TwoTowers
Tool Driver / Integrated Kernel
CWB−NC
Functional Verification
MarCA
Performance Evaluation

Figure 1.1 TwoTowers builds on CWB-NC and MarCA.

The purpose of TwoTowers is similar to that of other stochastically timed process algebra based tools, such as the PEPA Workbench [11] and the TIPP-tool [14]. However, besides the difference in the expressive power of the algebras used by these tools [5], the underlying philosophy is quite different. TwoTowers profits from two well-known software tools to carry out the functional and performance analysis. This is advantageous both from the point of view of the implementors and from the point of view of the users. We needed not to write code for implementing the analysis routines, thereby making the development of TwoTowers itself easier and faster, and users are provided with a wide range of automated techniques helping them during the study of systems.

The remainder of the paper is organized as follows. In the second section we present the language of TwoTowers, i.e. $\mathrm{EMPA_r}$. In the third section we present the architecture of TwoTowers. In the fourth section we describe our use of the tool to study the Lehmann-Rabin randomized distributed algorithm for the dining philosopher problem. Finally, in the fifth section we report some concluding remarks about related work and future extensions of TwoTowers.

THE LANGUAGE OF TWOTOWERS: EMPA$_{\rm R}$

The building blocks of EMPA$_{\rm r}$ [1] are *actions*. An action is a quadruple $<a, \tilde{\lambda}, \tilde{r}_1, \tilde{r}_2>$ where:

- a is the *type* of the action. Types divide actions into *external* and *internal* (denoted by action type τ) depending on whether they can be seen by an external observer or not.
- $\tilde{\lambda}$ is the *rate* of the action, i.e. a concise way to represent the random variable specifying its duration. Based on rates, we have the following classification:
  - *Exponentially timed actions* are actions whose rate is a positive real number. Such a number is interpreted as the parameter of the exponentially distributed random variable specifying the duration of the action.
  - *Immediate actions* are actions whose rate, denoted by $\infty_{l,w}$, is infinite. Such actions have duration zero and each of them is given a *priority level* $l \in \mathbb{N}_+$ and a *weight* $w \in \mathbb{R}_+$, useful for expressing prioritized and probabilistic choices respectively.
  - *Passive actions* are actions whose rate, denoted by $*$, is undefined. The duration of a passive action is fixed only by synchronizing it with a nonpassive action of the same type.

  The restriction to exponentially distributed durations implies that the semantics can be defined in the interleaving style, thanks to the memoryless property of exponential distributions, and that the underlying performance models are Markov chains.
- $\tilde{r}_1$ and $\tilde{r}_2$ are, respectively, the *yield reward* and the *bonus reward* of the action [15]. Both of them are included within actions to allow for a high level specification of the performance measures of interest. The former expresses the speed at which gain is accumulated at the state executing the action, whereas the latter expresses the gain awarded upon executing the action. Performance measures are computed as weighted sums of state probabilities and transition frequencies, where rewards are used as weights.

We denote by

$$Act_{\rm r} = \{<a, \tilde{\lambda}, \tilde{r}_1, \tilde{r}_2> \in AType \times ARate \times AYReward \times ABReward \mid \\ \tilde{\lambda} = * \Longleftrightarrow \tilde{r}_1 = \tilde{r}_2 = *\}$$

the set of actions, where $AType$ is the set of types, $ARate = \mathbb{R}_+ \cup Inf \cup \{*\}$, with $Inf = \{\infty_{l,w} \mid l \in \mathbb{N}_+ \wedge w \in \mathbb{R}_+\}$ is the set of rates, $AYReward = \mathbb{R} \cup \{*\}$ is the set of yield rewards, and $ABReward = \mathbb{R} \cup \{*\}$ is the set of bonus rewards.

Let $Const$ be a set of *constants* and $ARFun = \{\varphi : AType \longrightarrow AType \mid \varphi(\tau) = \tau \wedge \varphi(AType - \{\tau\}) \subseteq AType - \{\tau\}\}$ be a set of *action relabeling functions*. The set $\mathcal{L}_{\rm r}$ of *process terms* of EMPA$_{\rm r}$ is generated by the following syntax

$$E ::= \underline{0} \mid <a, \tilde{\lambda}, \tilde{r}_1, \tilde{r}_2>.E \mid E/L \mid E[\varphi] \mid E + E \mid E \parallel_S E \mid A$$

where $L, S \subseteq \mathit{AType} - \{\tau\}$ and $A \in \mathit{Const}$. We denote by $\mathcal{G}_{\rm r}$ the set of guarded and closed terms of $\mathcal{L}_{\rm r}$.

The *null term* "$\underline{0}$" represents a termination or deadlocked state. The *prefix operator* "$<a, \tilde{\lambda}, \tilde{r}_1, \tilde{r}_2>.\_$" denotes the sequential composition of an action and a term. The *functional abstraction operator* "$\_/L$" hides the actions whose type belongs to L by changing their type to τ. The *functional relabeling operator* "$\_[\varphi]$" changes the types of actions according to φ. The *alternative composition operator* "$\_ + \_$" expresses a choice between two terms: the choice is resolved according to the *race policy* whenever exponentially timed actions are involved (the fastest one succeeds) or according to the *preselection policy* whenever immediate actions are involved (only the actions having the highest priority level are executable and each of these is given an execution probability proportional to its own weight), while the choice is purely *nondeterministic* whenever passive actions are involved. The *parallel composition operator* "$\_ \parallel_S \_$" expresses the concurrent execution of two terms according to the following synchronization discipline: two actions can synchronize if and only if they have the same type belonging to S and at most one of them is not passive. Finally, constants together with their corresponding defining equations of the form $A \stackrel{\Delta}{=} E$ allow for recursion.

Given a term $E \in \mathcal{G}_{\rm r}$, its *integrated interleaving semantics* $\mathcal{I}_{\rm r}[\![E]\!]$ is represented as a transition system with action-labeled edges. Because of the presence of actions with different priorities, as well as the fact that immediate actions take precedence over exponentially timed ones due to the race policy, the integrated interleaving model is generated by a two-layer semantics: first all the potential moves are derived, then those with lower priority are discarded. The states of $\mathcal{I}_{\rm r}[\![E]\!]$ are divided into *tangible*, *vanishing*, *open*, and *absorbing* depending on whether they have outgoing exponentially timed transitions, outgoing immediate transitions, outgoing passive transitions, or no outgoing transitions. $E \in \mathcal{G}_{\rm r}$ is said to be *performance closed* if $\mathcal{I}_{\rm r}[\![E]\!]$ does not contain passive transitions; we denote by $\mathcal{E}_{\rm r}$ the set of performance closed terms of $\mathcal{G}_{\rm r}$.

From the integrated interleaving semantic model, two projected semantic models can be obtained. The *functional semantics* of $E \in \mathcal{G}_{\rm r}$ is still represented as a transition system $\mathcal{F}_{\rm r}[\![E]\!]$ now labeled by action types only, derived from $\mathcal{I}_{\rm r}[\![E]\!]$ by dropping action rates and rewards. The *performance semantics* of $E \in \mathcal{E}_{\rm r}$ is represented as a homogeneous Markov reward chain basically derived from $\mathcal{I}_{\rm r}[\![E]\!]$ by dropping action types and by lifting yield rewards from transitions to states: the yield reward earned by a state is the sum of the yield rewards earned by its outgoing transitions. To be more precise, the performance model is given by a homogeneous continuous time Markov reward chain or a homogeneous discrete time Markov reward chain depending on whether $\mathcal{I}_{\rm r}[\![E]\!]$ contains only exponentially timed or immediate transitions. If both kinds of transition coexist, the performance model is a homogeneous continuous time Markov reward chain built by applying to $\mathcal{I}_{\rm r}[\![E]\!]$ a suitable graph reduction rule, which eliminates vanishing states together with their outgoing immediate transitions and modifies rates of transitions entering such states accordingly.

For a more formal presentation of EMPA$_{\rm r}$ semantics, the interested reader is referred to [1, 2].

ARCHITECTURE OF TWOTOWERS

The architecture of TwoTowers is depicted in Fig. 1.2. TwoTowers is composed of a graphical user interface, a tool driver, an integrated kernel, a functional kernel, and a performance kernel.

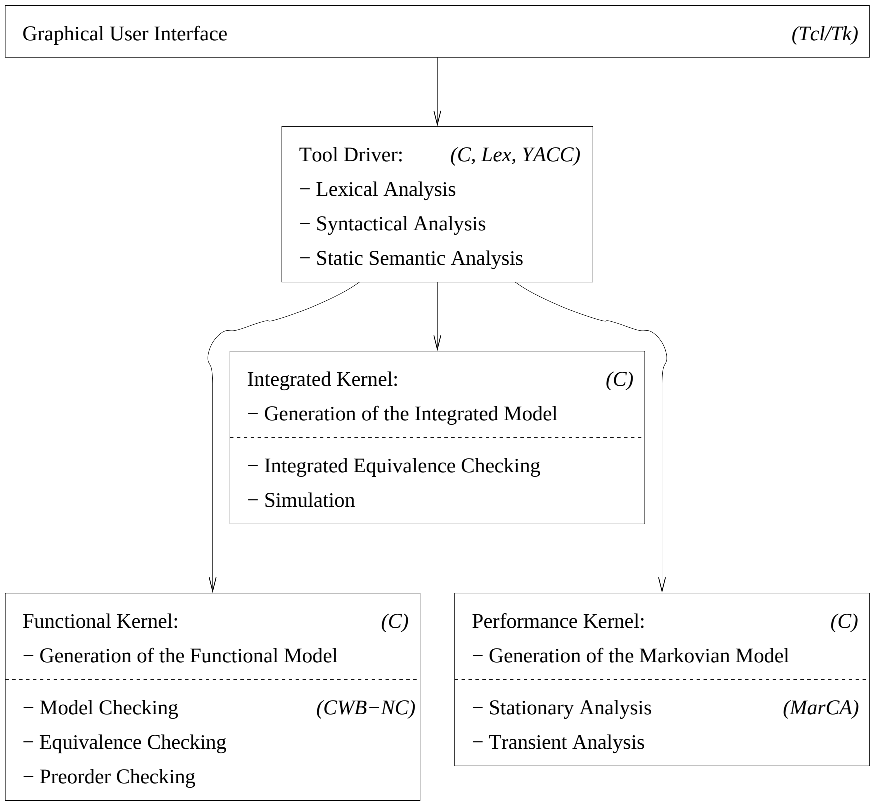

Figure 1.2 Architecture of TwoTowers.

The graphical user interface is written in Tcl/Tk [20] and allows the user to edit EMPA$_r$ specifications, compile them, and run the various analysis routines.

The tool driver, which is written in C [17] and uses Lex [19] and YACC [16], includes routines for parsing EMPA$_r$ specifications and performing lexical, syntactic and static semantic (closure, guardedness, finiteness) checks on the specifications. If there are errors, a message indicates the number of errors and a file is generated pinpointing the errors in the specification source file. When there are no errors, the syntax tree of the specification is produced for analysis by the three kernels.

The purpose of the integrated kernel is to perform those analyses that require both functional and performance information and therefore cannot be performed by factoring out information to be passed to either the functional or performance kernels. The integrated kernel, which is implemented in C, contains the routines to generate the integrated interleaving semantic model of correct EMPA$_\mathrm{r}$ specifications. The model generation employs the transition caching strategy proposed in [8] to store certain semantic information during model construction thereby avoiding recomputation of this information and saving time and memory. After generating a model the tool reports the number of states (classified as tangible, vanishing, open, or absorbing) as well as the number of transitions (classified as observable or invisible according to their types, and as exponentially timed, immediate, or passive according to their rates). Moreover, the integrated kernel contains a routine to check two EMPA$_\mathrm{r}$ specifications for strong extended Markovian reward bisimulation equivalence [1], as well as a facility for simulating an EMPA$_\mathrm{r}$ specification which supports the derivation of performance measures according to the method of independent replications [22].

The functional kernel, which is written in C, generates the functional semantic model of correct EMPA$_\mathrm{r}$ specifications. The analysis of the labeled transition systems as well as the terms themselves is then performed by a version of the Concurrency Workbench of North Carolina (CWB-NC) [9] that was retargeted for EMPA$_\mathrm{r}$ using the Process Algebra Compiler of North Carolina (PAC-NC) [8]. The functional semantics of EMPA$_\mathrm{r}$ along with a syntactic description of EMPA$_\mathrm{r}$ were encoded as input to the PAC-NC which then produced the code to retarget the CWB-NC to EMPA$_\mathrm{r}$. The TwoTowers functional kernel therefore comprises all the analysis capabilities of the CWB-NC including interactive simulation of systems, reachability analysis to check safety properties, model checking in the μ-calculus or CTL, equivalence checking, and preorder checking.

Finally, the performance kernel, which is written in C, generates the Markovian semantic model of correct performance closed EMPA$_\mathrm{r}$ specifications. The analysis of the Markov reward chains is then carried out by means of the Markov Chain Analyzer (MarCA) [21]: transient and stationary probability distributions are calculated, then performance indexes are derived using the rewards expressed in the specifications themselves.

ANALYSIS OF A DISTRIBUTED ALGORITHM

In this section we describe our use of TwoTowers to analyze the Lehmann-Rabin randomized distributed algorithm for the dining philosopher problem, which is characterized as follows. Suppose there are n philosophers P_i, $0 \leq i \leq n-1$, sitting at a round table each with a plate in front of him/her. Furthermore, let there also be n chopsticks C_i, $0 \leq i \leq n-1$, each shared by two neighbor philosophers and used to get the rice at the center of the table. Given that philosophers can be thought of as concurrent processes sharing resources represented by the chopsticks, the mutual exclusive use of the chopsticks must be guaranteed in such a way that no adjacent philosophers are simultaneously eating and deadlock does not occur.

A fair and elegant solution to this problem has been proposed in [18]. The idea is that P_i flips a fair coin to choose between C_i and C_{i+1}, gets the chosen chopstick as soon as it becomes free, and gets the other chopstick if it is free. If the second chopstick is not available, the philosopher replaces the first one and repeats the procedure. If we denote by "$\_ +_n \_$" ("$\_ -_n \_$") the addition (subtraction) modulo n, and let $think_i$ be the action type "P_i is thinking", eat_i be "P_i is eating", pu_i be "C_i is being picked up", and pd_i be "C_i is being put down", this algorithm can be modeled as follows with EMPA$_\mathrm{r}$:

$$\begin{array}{rcl}
\mathit{DP}_n & \equiv & (P_0 \parallel_\emptyset \dots \parallel_\emptyset P_{n-1}) \parallel_{\{pu_i, pd_i \mid 0 \le i \le n-1\}} (C_0 \parallel_\emptyset \dots \parallel_\emptyset C_{n-1}) \\
P_i & \stackrel{\Delta}{=} & <\mathit{think}_i, \lambda_i, 0, 0>.P'_i \\
P'_i & \stackrel{\Delta}{=} & <\tau, \infty_{1,1/2}, 0, 0>.<pu_i, \infty_{1,1}, 0, 0>.(<pu_{i+_n 1}, \infty_{2,1}, 0, 0>.P''_i + \\
 & & \hfill <pd_i, \infty_{1,1}, 0, 0>.P'_i) + \\
 & & <\tau, \infty_{1,1/2}, 0, 0>.<pu_{i+_n 1}, \infty_{1,1}, 0, 0>.(<pu_i, \infty_{2,1}, 0, 0>.P''_i + \\
 & & \hfill <pd_{i+_n 1}, \infty_{1,1}, 0, 0>.P'_i) \\
P''_i & \stackrel{\Delta}{=} & <\mathit{eat}_i, \mu_i, 1, 0>.<pd_i, \infty_{1,1}, 0, 0>.<pd_{i+_n 1}, \infty_{1,1}, 0, 0>.P_i \\
C_i & \stackrel{\Delta}{=} & <pu_i, *, *, *>.<pd_i, *, *, *>.C_i
\end{array}$$

where the τ actions correspond to coin flips with the weights reflecting the respective probabilistic outcomes. It is worth noting that only actions with type think_i or eat_i are relevant from the performance viewpoint and so they have been given an exponential timing, whereas the time it takes P_i to flip the coin or to pick up or put down a chopstick can be neglected so the corresponding actions are immediate. We also point out that priority levels of immediate actions have proven to be quite helpful in the description of the choice between getting the second chopstick and releasing the first one in order to model the fact that, whenever the second chopstick is available, the philosopher does pick it up. This is achieved in the subterms of term P'_i enclosed in parentheses by assigning priority level 2 to the action with type pu and priority level 1 to the action with type pd.

Now, we apply TwoTowers to the algebraic description of the algorithm in order to automatically verify that no adjacent philosophers be simultaneously eating and deadlock be avoided (functional properties ensuring the correctness of the algorithm), and to automatically assess the mean number of philosophers who are simultaneously eating (a performance index reflecting the degree of parallelism of the algorithm). To verify the two functional properties, we can resort to model checking by proving that every state satisfies respectively

$$\phi_1 = \bigwedge_{i=0}^{n-1} \neg(\langle \mathit{eat}_i \rangle \mathit{tt} \wedge (\langle \mathit{eat}_{i+_n 1} \rangle \mathit{tt} \vee \langle \mathit{eat}_{i-_n 1} \rangle \mathit{tt}))$$

and

$$\phi_2 = \langle - \rangle \mathit{tt}$$

The mean number ν of philosophers who are simultaneously eating is instead obtained by summing up the steady state probabilities of any state of the underlying Markov reward chain multiplied by the number of philosophers eating in that state, which is derived by yield reward 1 assigned to action having type eat_i in term P''_i.

In Table 1.1 we summarize the results we have obtained by varying the number n of philosophers from 3 to 6, where $\lambda_i = 4$ and $\mu_i = 0.75$ for any $0 \le i \le n-1$. The columns show, respectively, the number of philosophers, the number of states of the integrated interleaving semantic model, the number of tangible states (which coincides with the number of states of the Markov reward chain), the number of vanishing states, the number of transitions, the number of exponentially timed transitions, the number of immediate transitions, the absence of adjacent philosophers simultaneously eating

Table 1.1 Results of the analysis of the Lehmann-Rabin algorithm.

n	*states*	*ts*	*vs*	*transitions*	*ett*	*it*	ϕ_1	ϕ_2	ν
3	109	13	96	147	27	120	yes	yes	0.999084
4	387	35	352	620	100	520	yes	yes	1.758340
5	1101	81	1020	1655	285	1370	yes	yes	1.975830
6	3199	199	3000	5070	846	4224	yes	yes	2.530878

(verified by means of CWB-NC), the absence of deadlock (verified by means of CWB-NC), and the mean number of philosophers who are simultaneously eating (computed by means of MarCA).

CONCLUSION

In this paper we have presented TwoTowers, a tool based on the stochastically timed reward process algebra $\mathrm{EMPA_r}$ for the analysis of functional and performance properties of concurrent systems. TwoTowers generates the semantic models of algebraic specifications; then the task of studying such models is transferred to two already existing tools: CWB-NC for functional verification, MarCA for performance evaluation. Relying on existing tools is advantageous from the point of view of both implementors and users. We needed not to write code for implementing the analysis routines, thereby making the development of TwoTowers easier and faster, and users are provided with a wider range of automated techniques.

In this paper we have applied TwoTowers to the Lehmann-Rabin randomized distributed algorithm. More complex systems have been analyzed by means of TwoTowers, such as the alternating bit protocol [4], CSMA/CD [2], ATM switches supporting different services [2], and an adaptive mechanism for trasmitting packetized audio over the Internet [6]. When dealing with larger state spaces, suitable techniques should be employed to reduce the cost of the analysis:

- As far as functional verification is concerned, TwoTowers provides the user with the techniques implemented in the CWB-NC, such as on-the-fly model checking and compositional minimization.

- From the performance evaluation standpoint, all the techniques implemented in MarCA are available in TwoTowers, together with the support for simulative analysis which allows infinite state systems to be handled. Moreover, in the future capabilities for the detection at the syntax level of product form solutions (see e.g. [12]) shall be added.

- The planned implementation of the second phase of the approach proposed in [4] should help in the analysis, because this would be conducted on truly concurrent models given by stochastically timed Petri nets, instead of interleaving models.

Acknowledgments

This research has been partially funded by Italian MURST and CNR Progetto Strategico "Modelli e Metodi per la Matematica e l'Ingegneria", NSF Young Investigator Award CCR-9257963, NSF Award CCR-9403807, NSF Award INT-9603441, AFOSR Grant F49620-95-1-0508, and NSF Grant CCR-9413309.

References

[1] Bernardo, M. (1997) An Algebra-Based Method to Associate Rewards with EMPA Terms, in *Proc. of the 24th Int. Coll. on Automata, Languages and Programming (ICALP '97). Lecture Notes in Computer Science*, **1256**, 358–368.

[2] Bernardo, M. (1998) Formal Specification of Performance Measures for Process Algebra Models of Concurrent Systems. Technical Report UBLCS-98-08, University of Bologna (Italy).

[3] Bernardo, M., Donatiello, L. and Gorrieri, R. (1994) Integrated Analysis of Concurrent Distributed Systems using Markovian Process Algebra, in *Proc. of the 7th Int. Conf. on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE '94)*. Chapman & Hall, 455-457.

[4] Bernardo, M., Donatiello, L. and Gorrieri, R. (1998) A Formal Approach to the Integration of Performance Aspects in the Modeling and Analysis of Concurrent Systems, to appear in *Information and Computation*.

[5] Bernardo, M. and Gorrieri, R. (1998) A Tutorial on EMPA: A Theory of Concurrent Processes with Nondeterminism, Priorities, Probabilities and Time, *Theoretical Computer Science*, **202**, 1–54.

[6] Bernardo, M., Gorrieri, R. and Roccetti, M. (1998) Formal Performance Modeling and Evaluation of an Adaptive Mechanism for Packetized Audio over the Internet. Technical Report UBLCS-98-09, University of Bologna (Italy).

[7] Bochmann, G.V. and Vaucher, J. (1988) Adding Performance Aspects to Specification Languages, in *Proc. of the 8th Int. Conf. on Protocol Specification, Testing and Verification (PSTV VIII)*. North Holland, 19–31.

[8] Cleaveland, W.R., Madelaine, E. and Sims, S. (1995) A Front-End Generator for Verification Tools, in *Proc. of the 1st Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '95). Lecture Notes in Computer Science*, **1019**, 153–173.

[9] Cleaveland, W.R. and Sims, S. (1996) The NCSU Concurrency Workbench, in *Proc. of the 8th Int. Conf. on Computer Aided Verification (CAV '96). Lecture Notes in Computer Science*, **1102**, 394–397.

[10] Ferrari, D. (1986) Considerations on the Insularity of Performance Evaluation. *IEEE Trans. on Software Engineering*, **12**, 678–683.

[11] Gilmore, S. and Hillston, J. (1994) The PEPA Workbench: A Tool to Support a Process Algebra-based Approach to Performance Modelling, in *Proc. of the*

7th Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation. *Lecture Notes in Computer Science*, **794**, 353–368.

[12] Harrison, P.G. and Hillston, J. (1995) Exploiting Quasi-Reversible Structures in Markovian Process Algebra Models. *Computer Journal*, **38**, 510–520.

[13] Harvey, C. (1986) Performance Engineering as an Integral Part of System Design. *BT Technology Journal*, **4**, 143–147.

[14] Hermanns, H., Mertsiotakis, V. and Rettelbach, M. (1996) A Construction and Analysis Tool Based on the Stochastic Process Algebra TIPP, in *Proc. of the 2nd Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '96)*. *Lecture Notes in Computer Science*, **1055**, 427–430.

[15] Howard, R.A. (1971) *Dynamic Probabilistic Systems*. John Wiley & Sons.

[16] Johnson, S.C. (1975) *YACC - Yet Another Compiler Compiler*. Computing Science Technical Report 32, AT&T Bell Labs.

[17] Kernighan, B.W. and Ritchie, D.M. (1988) *The C Programming Language*. Prentice Hall.

[18] Lehmann, D. and Rabin, M. (1981) On the Advantage of Free Choice: A Symmetric and Fully Distributed Solution to the Dining Philosophers Problem, in *Proc. of the 8th Symp. on Principles of Programming Languages (POPL '81)*. ACM Press, 133–138.

[19] Lesk, M.E. (1975) *Lex - A Lexical Analyzer Generator*. Computing Science Technical Report 39, AT&T Bell Labs.

[20] Ousterhout, J.K. (1994) *Tcl and the Tk Toolkit*. Addison-Wesley.

[21] Stewart, W.J. (1994) *Introduction to the Numerical Solution of Markov Chains*. Princeton University Press.

[22] Welch, P.D. (1983) The Statistical Analysis of Simulation Results, in *Computer Performance Modeling Handbook* (ed. S.S. Lavenberg), Academic Press, 267–329

[23] Yemini, Y. and Kurose, J. (1982) Towards the Unification of the Functional and Performance Analysis of Protocols, or Is the Alternating-Bit Protocol Really Correct?, in *Proc. of the 2nd Int. Conf. on Protocol Specification, Testing and Verification (PSTV II)*. North Holland.